CS 171 Computer Programming 1

Winter 2016

Lab 4 – Loops

NAME: LAB:

Question 1: Stalking Pi (5pts)

Many problems in mathematics, science, and engineering do not have a solution with a simple formula. In these cases, computer scientists design algorithms to solve problems either exactly or approximately. In the latter case, one may be able to design and implement an *iterative* procedure that *converges* to the correct answer. For example, consider the following formula:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

Each time we add another term into the sum, the *partial sum* gets closer and closer to the actual value of Pi. For these iterative techniques, one has to decide how long to continue the process before the answer is "close enough." There are two basic stopping criteria: either two successive computations yield little improvement, or the computation has gone on long enough that we no longer wish to continue. Note that in the second case, we have no guarantee that the results are any good.

Although it might seem appealing, it is impossible to actually sum forever to find a value for Pi. Instead we must choose a level of accuracy which will satisfy us, and then stop summing once that level of accuracy is reached.

Download and examine the code for pi.cpp. Before running the program, answer in your lab solution “under what conditions will this iteration stop?”

Answers: The program will stop once the count of iterations is higher than 2000 ***or*** *the accuracy of the result is in the ten millionths decimal spot or better.*

Question 2: First Run (3pts)

Now run `pi.cpp (of course you likely will have to make a project for it, compile, etc...)`. Why did the program terminate? Answer in the lab solution whether it terminated due to desired accuracy or due to the maximum number of iterations being reached.

Answer: It terminates as a result of the max number of iterations being reached.

Question 3: Length of First Run (3pts)

Grab a watch! Approximately how long did the program take to execute? Of course this is dependent on your system. In the answer sheet record this time and your guesses on how long you expect the program to take to perform 20,000 iterations and 200,000 iterations (run them if you dare!).

Answer: 4.61 seconds for 2000, 46.1 second for 20,000, 461 seconds for 200,000.

Question 4: More Selective Output (5pts)

The program (as you hopefully noticed) prints out the current guess of Pi on each iteration. It is probably not necessary (OK, *definitely not necessary*) to print this out on every iteration (it takes a long time, relative to other operations, to print stuff to the command line).

Modify the code so that the results are only printed after every 100 iterations. (HINT: use the `count` variable.) (Another hint: you don't need to construct an inner loop or anything, just use `count`.)

Copy your modified code into your answer sheet.

Answer:

```
if (count % 100 == 0)
{
	cout << "Current guess for PI ("
		<< count << " iterations) = "
		<< setprecision(10)
		<< 4 * newPartialSum << endl;
	cout << "Enter anything to continue the program" << endl;
```

```
        cin.get();
}
```

Question 5: More Iterations (5pts)

As mentioned, your code should have run a lot faster without all that output. Your final task is to determine exactly how many iterations your program needs before the result is sufficiently accurate. That is, keep increasing the maximum number of iterations and running the program until it quits because of accuracy rather than reaching the maximum. Don't cheat and just increase the number of possible iterations to billions. We want to find the *minimum* number of iterations necessary to allow for termination due to accuracy. You may decide to print out your results less frequently than every 100 iterations, also.

At the end (once you have your program terminating due to accuracy rather than iterations) include in your answer box the minimum necessary number of iterations so that your program terminates due to desired accuracy. Also record the final computation of Pi.

Answer: the minimum number of iterations is 5,000,001, final result is: 3.141592854

Question 6: Falling to the Moon (5pts)

Over the course of the next several questions we will be developing a simulation of landing a lunar module on the surface of the moon. Okay, let's be honest. It's a lunar lander game.

Let's start with the motion of our lander simply falling to the moon's surface. All we need to know to simulate this free fall is the rate of acceleration due to gravity. Because the gravity of the moon is only 1/6 that here on Earth, an object falling on the moon will accelerate at the rate of -1.63 m/s^2.

Your first task for this is to write a program (from scratch) that prompts the user for an initial altitude and an initial velocity. It then updates the altitude and the velocity after a second has passed, and prints out the new values with two decimal values after the decimal place.

To do this, we start with an altitude of a meters and a velocity of `v` meter per second. One second later, the altitude will be `a+v`. Similarly, if we start with a velocity of `v` meters per second and an acceleration due to gravity of `g` meters per second squared, one second later the velocity will be `v+g`.

Once you have your program running, copy your code to the answer sheet.

Answer:

```
#include <iostream>

#include <iomanip>

#include <cmath>

using namespace std;

int main(void)

{

	double gravity = 0;

	double const constGravity = 1.36;
```

```
	cout << "please enter the initial altitude in terms of meters" << endl;
	double altitude;
	cin >> altitude;
	cout << "please enter the initial velocity in terms of meters per second" << endl;
	double velocity;
	cin >> velocity;
	int second = 0;
	altitude = altitude - velocity;
	while ((altitude - (constGravity + gravity)) >= 0)
	{

		altitude = altitude - (constGravity + gravity);
		cout << "distence from ground: " << altitude << " meters" << endl;
		cout << "seconds: " << second << endl;
		cout << "gravity: " << gravity << endl;
		cout << "Enter anything to continue the program" << endl;
		cin.get();
		second++;
		gravity = (1.36 * second);
	}
}
```

Question 7: Go to the Ground (5pts)

Modify your program to *repeat* the "one second step" *until* the lander hits the ground (or technically, goes "underground"), printing the altitude and velocity at each one-second interval.

Copy your new code to the answer sheet.

Answer: I had already done this for the sake of debugging

```
#include <iostream>
#include <iomanip>
```

```
#include <cmath>
using namespace std;
int main(void)
{
	double gravity = 0;
	double const constGravity = 1.36;
	cout << "please enter the initial altitude in terms of meters" << endl;
	double altitude;
	cin >> altitude;
	cout << "please enter the initial velocity in terms of meters per second" << endl;
	double velocity;
	cin >> velocity;
	int second = 0;
	altitude = altitude - velocity;
	while ((altitude - (constGravity + gravity)) >= 0)
	{

		altitude = altitude - (constGravity + gravity);
		cout << "distence from ground: " << altitude << " meters" << endl;
		cout << "seconds: " << second << endl;
		cout << "gravity: " << gravity << endl;
		cout << "Enter anything to continue the program" << endl;
		cin.get();
		second++;
		gravity = (1.36 * second);
	}
}
```

Question 8: Facts on the Ground (2pts)

Based on your code from the previous question, in the answer sheet answer the following questions by running your code on some sample inputs:

1. Is the lander traveling faster when it hits the ground than when it starts?
2. If your initial velocity is 0, does the final velocity depend on the initial altitude?

Answers:

1. yes as a result of the gravity
2. yes as long as the gravity is a const

Question 9: Safe Landing or Crash? (3pts)

Now we want to add a test to see if we are moving faster than $-2\mathrm{m/s}$ when we touch down. If it is, print that we have crashed, otherwise print a message saying that we have landed safely.

Copy your new code to your answer sheet.

Answer:

```
if ((velocity + constGravity + gravity) > 2)
{
        cout << "you have crashed" << endl;
}
else
{
        cout << "you have landed safley" << endl;
}
```

Question 10: Safe Height (1pt)

By running your code form the previous problem on several tests write in your answer sheet how low to the ground (just test integer heights, although your program should support floating point heights) do we have to be with an initial velocity value of 0 to have a safe landing?

Answer:

1 meter

Question 11: Controlling Our Speed (3pts)

Our next step is to allow us to burn fuel to slow our descent. At the end of each iteration (actually before adding the gravity to the velocity), ask the user for a number of fuel units to burn during that second. If we burn `f` units of fuel, our acceleration during that second will be `-1.63+0.1f` (as opposed to our normal `-1.63`).

Copy your code to your answer sheet

Answer:

```
while ((altitude - (constGravity + gravity)) >= 0)
{
        cout << "How much fuel do you want to burn for the next second" << endl;
        cin >> fuel;
        gravity = ((1.36 - (fuel*0.1)) * second);
        altitude = altitude - (constGravity + gravity);
        cout << "distence from ground: " << altitude << " meters" << endl;
```

```
        cout << "seconds: " << second << endl;
        cout << "gravity: " << gravity << endl;
        cout << "Enter anything to continue the program" << endl;
        cin.get();
        second++;

}
```

Question 12: Validating the Fuel Input (5pts)

Some questions worth asking ourselves include:

- Does your program allow for a negative fuel usage during a simulation interval?
- What would spending a negative amount of fuel mean? Is that realistic?

Hopefully you recognized that the answer to that question is no.

Your next step is to modify your program to validate the fuel amount input. If the user inputs a negative number, you should print an error message and prompt again.

After modifying and testing your code cut and paste it into your answer sheet.

Answer: without using functions or recursion I will only be able to prompt the user once and hope that they input the correct form of data.

```
        cout << "How much fuel do you want to burn for the next second" << endl;
        cin >> fuel;
        if (fuel < 0){
               cout << "please enter a positive value for fuel now" << endl;
               cin >> fuel;
        }
```

Question 13: Extra Credit – Fuel Limits (3pts EC)

For extra credit, you should add a limit on the amount of fuel available to burn. Therefore after asking for the initial velocity also ask for the initial fuel amount. If the user exhausts the amount of fuel in your lander, your simulation should continue until then lander hits the ground under only the influence of gravity.

Cut and paste your final program into the answer sheet.

Answer:

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main(void)
{
	double gravity = 0;
	double const constGravity = 1.36;
	cout << "please enter the initial altitude in terms of meters" << endl;
	double altitude;
	cin >> altitude;
	cout << "please enter the initial velocity in terms of meters per second" << endl;
	double velocity;
	cin >> velocity;
	int second = 0;
	altitude = altitude - velocity;
	double fuel;
	double fuelAmount = 100;
	while ((altitude - (constGravity + gravity)) >= 0)
	{
		cout << "How much fuel do you want to burn for the next second" << endl;
		cin >> fuel;
		if (fuel < 0){
			cout << "please enter a positive value for fuel now" << endl;
```

```
                cin >> fuel;
        }
        fuelAmount = fuelAmount - fuel;
        gravity = ((1.36 - (fuel*0.1)) * second);
        altitude = altitude - (constGravity + gravity);
        cout << "distence from ground: " << altitude << " meters" << endl;
        cout << "fuel left: " << fuelAmount << endl;
        second++;

    }
    if ((velocity + constGravity + gravity) > 2)
    {
        cout << "you have crashed" << endl;
    }
    else
    {
        cout << "you have landed safley" << endl;
    }
    cin.ignore(100);
    cout << "Enter anything to continue the program" << endl;
    cin.get();
}
```